Round-optimal Contributory Conference Key Agreement

Colin Boyd and Juan Manuel González Nieto

Information Security Research Centre,
Queensland University of Technology, Brisbane, Australia
{boyd,juanma}@isrc.qut.edu.au

Abstract. Becker and Wille derived a lower bound of only one round for multi-party contributory key agreement protocols. Up until now no protocol meeting this bound has been proven secure. We present a protocol meeting the bound and prove it is secure in Bellare and Rogaway's model. The protocol is much more efficient than other conference key agreement protocols with provable security, but lacks forward secrecy.

1 Introduction

Communications efficiency is concerned with the number and length of messages that need to be sent and received during a protocol. As well as minimising the number of individual messages, it can be important to have as few *rounds* as possible in the protocol. One round includes all the messages that can be sent in parallel during the protocol. Protocols where the messages are independent of each other require fewer rounds than those where messages include fields received in previous protocol messages.

Most published key agreement protocols are based on Diffie-Hellman's famous key exchange protocol. A number of generalisations of the Diffie-Hellman protocol have been devised which allow many parties to agree jointly on a session key. With the exception of a recent protocol proposed by Joux for three parties in a special setting [19], all these generalisations require multiple rounds of communications in order to complete.

In 1998, Becker and Wille [3] derived several bounds on multi-party key agreement protocols. Amongst these was the bound on the number of rounds which is only one, no matter what is the number of users involved. A protocol that meets this bound would allow all messages to be sent simultaneously in one time unit, as long as parallel messages are possible. No Diffie-Hellman generalisation is able to meet this bound and Becker and Wille leave as an open question whether any contributory key agreement scheme can meet this bound.

The purpose of this paper is to describe a protocol which meets the bound of Becker and Wille. In addition we present a new proof of the security of the protocol under the assumption that standard secure cryptographic primitives exist for encryption and signature, and using ideal hash functions (random oracles). Although Becker and Wille considered only unauthenticated key agreement protocols, which are insecure against active adversaries, we provide a proven secure

authenticated key agreement which is secure in the usual understanding of key establishment protocols. Despite this extra security we still meet the bound. The protocol is simple and very efficient in comparison with previously published conference key agreement protocols. Its only significant limitation is that it does not provide forward secrecy. We regard the following as the main contributions.

- A new conference key agreement protocol is proven secure in the random oracle model.
- The computational requirements for the protocol are smaller than those for any existing provable secure conference key protocol.
- The first proven secure protocol that meets the bound of Becker and Wille for a single round protocol.

1.1 Related Work

Conference Key Agreement Most conference key agreement protocols are based on generalisations of Diffie and Hellman's famous key exchange protocol [15]. Examples include a set of protocols by Ingemarsson, Tang and Wong [18], a protocol by Burmester and Desmedt [14] and three protocols of Steiner, Tsudik and Waidner [23].

In their basic form none of these protocols provides authentication of the users so they do not protect against active attacks. Ateniese *et al.* [1, 2] propose two ways to extend one of the protocols of Steiner *et al.* to provide authenticated group key agreement. However, Pereira and Quisquater [21] have described a number of potential attacks, highlighting the need for ways to obtain greater assurance in the security of such protocols.

The protocol of Joux [19] is the only example currently known of a group key agreement protocol that can be run in a single round and still provide forward secrecy; however the protocol can only work with three parties. Joux's protocol works in elliptic curve groups and exploits pairings of group points.

Provable Security for Protocols An important direction in cryptographic protocol research was pioneered by Bellare and Rogaway in 1993 [7] when they published the first mathematical proof that a simple entity authentication protocol was secure. This work, which covered only the two-party case, was followed up with a paper on server-based protocols [4] and various authors have extended the same idea to include public-key based key transport [8], key agreement protocols [9], password-based protocols [6, 10], and conference key protocols [13, 11, 12].

The general approach is to produce a mathematical model that defines a protocol in which a powerful adversary plays a central role. The adversary essentially controls all the principals and can initiate protocol runs between any principals at any time. Insider attacks are modelled by allowing the adversary to corrupt any principals, and the adversary can also obtain previously used keys. Cryptographic algorithms are modelled in an idealised manner. Security of protocols is defined in terms of matching conversations (for authentication) and indistinguishability (for confidentiality of keys). The proofs follow the style

of most proofs in modern cryptography by reducing the security of the protocol to the security of some underlying primitive.

Up until now the only conference key protocols which carry a reduction proof are those of Bresson *et al.* [13, 11, 12]. Based on the generalised Diffie-Hellman protocols of Steiner *et al.* [23], these protocols are (relatively) computationally expensive. In addition they require a number of rounds equal to the number of principals in the conference. The advantage they have over the protocol examined in this paper is the provision of forward secrecy.

1.2 Protocols of Tzeng and Tzeng

Two new round-efficient conference key agreement protocols were presented by Tzeng and Tzeng [24]. Although they claim that their protocols can be completed in one round and are proven secure, we would like to point out the following limitations of their protocols.

- Their protocols require a session identifier to be known by all participating principals. Unless this session identifier is agreed beforehand their protocols cannot be completed in one round.
- Although they claim that their protocol provides authentication and does not leak information, they provide no reduction proof for a powerful adversary.

Like our protocol, the protocols of Tzeng and Tzeng do not provide forward secrecy. A feature of their protocols is a proof of knowledge that each party has been sent the same inputs. The purpose of these proofs is to provide fault detection and exclude principals who deviate from the protocol. However, such a proof is only useful on the strong assumption that the broadcast channel which they use provides integrity of all messages; otherwise a malicious insider can send different proofs to different principals.

The first protocol of Tzeng and Tzeng uses a conventional type of signature to provide authentication of signatures. However, their second protocol attempts to provide authentication implicitly by including the private key of the sender in the proof. Unfortunately this proof is not sound and consequently the protocol can be broken. The details are presented in Appendix A.

1.3 Outline of Paper

Sections 2 and 3 present the communications model and definitions of security which we use. These follow quite closely the definitions of Bellare and Rogaway [4] with modifications required for the special situation of conference keys. Section 4 presents our new protocol and explains the differences from other related protocols. Section 5 presents the proof of security.

2 Communications Model

We follow closely the model established by Bellare and Rogaway [7, 4] incorporating later updates [6]. In particular we use the later form of *partnering* which seems more suitable for the multi-party environment.

The adversary $\mathcal{A}$ is a probabilistic machine that controls all the communications that take place and does this by interacting with a set of *oracles*, each of which represents an instance of a principal in a specific protocol run. Each principal has an identifier U from a finite set $\{U_1, \ldots, U_n\}$. Oracle Π_U^s represents the actions of principal U in the protocol run indexed by integer s. The number of principals n is polynomial in the security parameter k. Each user has a long-lived key, obtained at the start of the protocol using a key distribution algorithm $\mathcal{G}_L$. Interactions with the adversary are called oracle *queries* and the list of allowed queries is summarised in Table 1. We now describe each one informally.

$\mathsf{Send}(U, s, m)$	Send message m to oracle Π_U^s
$\mathsf{Reveal}(U, s)$	Reveal session key (if any) accepted by Π_U^s
$\mathsf{Corrupt}(U, K)$	Reveal state of U and set long-term key of U to K
$\mathsf{Test}(U, s)$	Ask for test key to distinguish session key accepted by oracle Π_U^s

Table 1. Queries available to the adversary

$\mathsf{Send}(U, s, m)$ This query allows the adversary to make the principal U run the protocol normally. The oracle Π_U^s will return to the adversary the same next message that an honest principal U would if sent message m according to the conversation so far. (This includes the possibility that m not be of the expected format in which case Π_U^s may simply halt.) If Π_U^s accepts the session key or halts this is included in the response. The adversary can use this query to initiate a new protocol instance by sending a flag message $m = \mathsf{Initiator}$ or $m = \mathsf{Responder}$ indicating the role that the principal plays.

$\mathsf{Reveal}(U, s)$ This query models the adversary's ability to find session keys. If a session key K_s has previously been accepted by Π_U^s then it is returned to the adversary. An oracle can only accept a key once (of course a principal can accept many keys modelled in different oracles). An oracle is called *opened* if it has been the object of a Reveal query.

$\mathsf{Corrupt}(U, K)$ This query models insider attacks by the adversary. The query returns the oracle's internal state and sets the long-term key of U to be the value K chosen by the adversary. The adversary can then control the behaviour of U with Send queries. A principal is called *corrupted* if it has been the object of a $\mathsf{Corrupt}$ query.

$\mathsf{Test}(U, s)$ Once the oracle Π_U^s has accepted a session key K_s the adversary can attempt to distinguish it from a random key as the basis of determining security of the protocol. A random bit b is chosen; if $b = 0$ then K_s is returned while if $b = 1$ a random string is returned from the same distribution as session keys. This query is only asked once by the adversary.

3 Security

Definitions of security in the Bellare-Rogaway model depend on the notion of the *partner* oracles to any oracle being tested. The way of defining partner oracles has varied in different papers using the technique. In the more recent research partners have been defined by having the same session identifier (SID) which consists of a concatenation of the messages exchanged between the two. Partners must both have accepted the same session key and recognise each other as partners. Bresson *et al.* [13] defined a set of session IDs for an oracle so that oracles in the same session should share session IDs pairwise. However, since all messages in our protocol are broadcast we can expect all oracles in the same session to derive the same session ID. Therefore we define $SID(\Pi^s_U)$ as the concatenation of all (broadcast) messages that oracle Π^s_U has sent and received.

Definition 1. *A set of oracles are* partnered *if:*

- *they have accepted with the same session ID,*
- *they agree on the set of principals, and*
- *they agree on the initiator of the protocol.*

Note that an oracle only 'knows' which principals it is communicating with but not which instance of the principals is involved.

Definition 2. *An oracle Π^s_U is* fresh *at the end of its execution if:*

- *Π^s_U has accepted with set of partners Π^*;*
- *Π^s_U and all oracles in Π^* are unopened;*
- *All principals of oracles in Π^* (including U) are uncorrupted.*

The security of the protocol is defined by the following game played between the adversary and an infinite collection of oracles Π^s_U for $U \in \{U_1, \ldots, U_n\}$ and $s \in \mathbb{N}$. Firstly, long-lived keys are assigned to each user by running the key distribution algorithm $\mathcal{G}_L$ on input of the security parameter. Then, the adversary $\mathcal{A}(1^k)$ is run. $\mathcal{A}$ will interact with the oracles through the queries defined above. At some stage during the execution a Test query is performed by the adversary to a fresh oracle. The adversary may continue to make other queries and eventually outputs a bit b' and terminates. Success of the adversary $\mathcal{A}$ in this game is measured in terms of its *advantage* in distinguishing the session key of the Test query from a random key, i.e. its advantage in outputting $b' = b$. This advantage must be measured in terms of the security parameter k. If we define Good-Guess to be the event that $\mathcal{A}$ guesses correctly whether $b = 0$ or $b = 1$ then

$$\mathsf{Advantage}^{\mathcal{A}}(k) = 2 \cdot \Pr[\mathsf{Good\text{-}Guess}] - 1.$$

To define validity of a conference key agreement protocol, we use the concept of a *benign adversary* as an adversary that faithfully relays flows between participants [7].

Definition 3. *A protocol P is a* secure conference key agreement scheme *if the following two properties are satisfied:*

- Validity*: in the presence of a benign adversary partner oracles accept the same key.*
- Indistinguishability*: for every probabilistic polynomial time adversary $\mathcal{A}$,* $\mathsf{Advantage}^{\mathcal{A}}(k)$ *is negligible.*

In the literature [20] it is often stated that a requirement of any key establishment protocol is *key authentication*, which means that each principal should have assurance that no other party has possession of the session key. A superficial examination of the above definition (or any of the several related ones in papers using the Bellare-Rogaway model) indicates that key authentication is ignored since the definition only refers to the adversary gaining information about the session key and not the identity of principals holding the key. In reality key authentication is implicitly included through the notion of partnering. If an oracle should accept a session key that is shared with an unknown principal then that principal is *not* the partner of the accepting oracle and can therefore be opened by the adversary and so the adversary does indeed gain an advantage.

Security of a protocol is proved by finding a reduction to some well known computational problem whose intractability is assumed. For the new protocol that we present in this paper, this reduction is to the security of the underlying public key encryption and signature schemes. Thus, we require notions of secure encryption and signature which are by now quite standard.

3.1 Secure Encryption Schemes

Let k denote the security parameter. A *public-key encryption scheme* $\mathcal{PE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms.

- The *key generation* algorithm $\mathcal{K}$ is a probabilistic algorithm which, on input 1^k, outputs a pair (e, d) of matching public and private keys, respectively.
- The *encryption* algorithm $\mathcal{E}$ is a probabilistic algorithm which takes a public key e and a message m drawn from a message space M associated to e and returns a ciphertext c. This is denoted as $c \stackrel{R}{\leftarrow} \mathcal{E}_e(m)$.
- The *decryption* algorithm $\mathcal{D}$ is a deterministic algorithm which takes a private key d and a ciphertext c and returns the corresponding plaintext m. This is denoted as $m \leftarrow \mathcal{D}_d(m)$. We require that $\mathcal{D}_d(\mathcal{E}_e(m)) = m$ for every $(e, d) \leftarrow \mathcal{K}(1^k)$.

For security we use the standard definition of *semantic security* due to Goldwasser and Micali [16]. For any probabilistic polynomial time adversary $\mathcal{A}$, the security is defined in terms of the following game.

1. Choose a key pair $(e, d) \leftarrow \mathcal{K}(1^k)$.
2. Given e, the adversary outputs two messages of equal length $m_0, m_1 \in M$ of her choice.

3. Compute $c_b \stackrel{R}{\leftarrow} \mathcal{E}_e(m_b)$ where $b \stackrel{R}{\leftarrow} \{0,1\}$. The bit b is kept secret from the adversary.
4. The adversary is then given c_b and has to output a guess b' for b.

We define the advantage of the adversary playing the above game as $\mathsf{Advantage}^{\mathcal{A}}(k) = 2 \cdot \Pr[b' = b] - 1$. The encryption scheme $\mathcal{PE}$ is secure if the adversary's advantage is negligible.

3.2 Secure Signature Scheme

Let k be the security parameter. A *digital signature scheme* $\Sigma = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ consists of the following three algorithms.

- The *key generation algorithm* $\mathcal{K}$ is a probabilistic algorithm that takes as input 1^k and outputs a pair of matching keys (e, d). The string e is the (public) *verification key*, and d the corresponding (private) *signing key*.
- The *signing algorithm* $\mathcal{S}$ takes as input a signing key d and a plaintext message m and outputs a signature σ.
- The *verification algorithm* $\mathcal{V}$ takes as input a verification key e, a message m and a signature σ, and outputs 1 if the signature is valid, and 0 otherwise.

For a signature scheme to be secure we require that it be computationally impossible for any adversary to forge a signature on any message (*existential forgery*) even *under adaptive chosen-message attacks* [17].

4 The Protocol

We now define the protocol that we shall prove secure. All parameter choices depend on a security parameter k. The protocol that we analyse involves the set of n users, $\mathcal{U} = \{U_1, U_2, \ldots, U_n\}$. The protocol has associated a secure public key encryption scheme $\mathcal{PE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\mathcal{K}$ is the key generation algorithm, and $\mathcal{E}, \mathcal{D}$ are the encryption and decryption algorithms, respectively. The protocol also uses a secure signature scheme $\Sigma = (\overline{\mathcal{K}}, \mathcal{S}, \mathcal{V})$, with $\overline{\mathcal{K}}$ the key generation algorithm, $\mathcal{S}$ the signing algorithm and $\mathcal{V}$ the verification algorithm. The key distribution algorithm $\mathcal{G}_L$ assigns to each user U_i an encryption/decryption key pair $(e_i, d_i) \leftarrow \mathcal{K}(1^k)$ and a signing/verification key pair $(\overline{e}_i, \overline{d}_i) \leftarrow \overline{\mathcal{K}}(1^k)$. The key distribution algorithm $\mathcal{G}_L$ also provides each user with an authentic copy of the public keys of all other users.

Each user, U_i, chooses a *nonce* (a random value, N_i, of size k bits). One user, say U_1, will be distinguished and will send its value N_1 to each other user in an authenticated and confidential way. We call this distinguished user the *initiator* of the protocol and the other users the *responders*. In an implementation there is no need for the messages of U_1 to be sent before the other messages, so it is perfectly possible for all messages to be sent together in one round.

The responders only have to broadcast their nonces so that all users in $\mathcal{U}$ receive all the N_i values. U_1 will encrypt N_1 for each other user U_i using U_i's

public encryption key e_i. U_1 will then sign the encrypted values of N_1 together with the names of all users in the conference. Since this message is the same for every user it only needs to be formed and sent once in a broadcast to all users. The value of N_1 is sent to user U_i encrypted with that user's public key, e_i. Thus, the protocol has three stages, all of which broadcast a message; in some communications scenarios each broadcast constitutes $n-1$ messages.

1. $U_1 \rightarrow * : \mathcal{U}, \mathcal{S}_{d_1}(\mathcal{U}, \mathcal{E}_{e_2}(N_1), \mathcal{E}_{e_3}(N_1), \ldots, \mathcal{E}_{e_n}(N_1))$
2. $U_1 \rightarrow * : \mathcal{E}_{e_i}(N_1)$ for $i \leq 2 \leq n$
3. $U_i \rightarrow * : U_i, N_i$

$$K_{\mathcal{U}} = h(N_1||N_2||N_3 \ldots ||N_n)$$

Fig. 1. Protocol execution with a benign adversary

Figure 1 shows the message flows in a protocol run without any disruption from the adversary. In the communications model this corresponds to the situation in which the adversary is benign, i.e. simply passes messages between principals. The asterisk is used to denote broadcast messages. The conference key should then be defined as follows, where h is a one-way function which will be modelled as a random oracle in the proof.

$$K_{\mathcal{U}} = h(N_1||N_2||N_3 \ldots ||N_n) \tag{1}$$

Let us consider the computational requirements for each user. U_1 has to perform $n-1$ public key encryptions and generate one signature. The other $n-1$ users have only to check one signature and decrypt one message, so for them the computational requirements are the same as for the two user case. The computational burden of U_1 can be reduced substantially by careful choice of public key cryptosystem. The computations required are substantially less than in the proven secure generalised Diffie-Hellman protocols of Bresson *et al.* [13, 11, 12], which require U_i to perform $i+1$ exponentiations in addition to generating and verifying a signature.

In common with most conference key protocols, we provide no confirmation to principals that others principals have obtained the session key. It is not possible to be sure whether some participants have been 'excluded' by an adversary who cuts off their incoming communications. Providing such assurances seems difficult and expensive to achieve.

5 Security Proof

The proof follows that of Bellare and Rogaway [4]; differences include the number of entities involved and the different partnering function used. The validity of the protocol is straightforward to verify. Thus, it remains to prove that the protocol satisfies the indistinguishability requirement. The general idea of the security proof is to assume that the adversary can gain a non-negligible advantage in

distinguishing test keys, and use this to break the assumption about the security of the underlying encryption scheme or the signature scheme. Since the adversary relies on its oracles to run we simulate the oracles so that we can supply the answers to all the queries the adversary might ask.

In our protocol we assume that the principals involved in each conference are the same. We do *not* assume that the same principal acts as the initiator. The case where the set of principals is chosen dynamically is easily handled too. The effect on the security proof is to make the reduction less tight.

Following Bellare and Rogaway [4] we need to extend the definition of a secure encryption scheme to allow the adversary to obtain encryptions of the same plaintext under multiple different independent encryption keys. Such an adversary is termed a *multiple eavesdropper*. We can bound the advantage of a multiple eavesdropper by considering it as a special case of the multi-user setting analysed by Bellare *et al.* [5]. In their notation we have the case of $q_e = 1$, meaning that the eavesdropper can only obtain one encryption for each public key. Let r be the number of encryptions of the same plaintext message seen by a multiple eavesdropper. Specialising their main theorem gives the following.

Lemma 1 ([5]). *Suppose that an adversary has advantage at most $\epsilon(k)$ for encryption scheme $\mathcal{PE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Then a multiple eavesdropper has advantage not more than $r \cdot \epsilon(k)$.*

We follow Bresson *et al.* [13] in dividing the proof into two cases. Firstly we consider the case in which the adversary $\mathcal{A}$ gains her advantage by forging a signature with respect to some user's signing key. In this case we construct a simple signature forging algorithm $\mathcal{F}$ against Σ that uses $\mathcal{A}$. In the second case, $\mathcal{A}$ gains her advantage without forging a signature. Then, we can construct an algorithm $\mathcal{X}$ that uses $\mathcal{A}$ against the security of the encryption algorithm.

5.1 Signature forger

Assume that $\mathcal{A}$ gains an advantage by forging a signature for some principal. We use $\mathcal{A}$ to construct a forger $\mathcal{F}$ for the signature scheme Σ. When $\mathcal{F}$ runs, it receives a public key $\overline{e}$ generated by $\overline{\mathcal{K}}(1^k)$ and access to a signing oracle for the corresponding signing key. The objective of $\mathcal{F}$ is to output a valid signature for a message which was not previously asked of the signing oracle.

In order to obtain the forgery $\mathcal{F}$ runs $\mathcal{A}$ with the following setting. Firstly, $\mathcal{F}$ chooses at random a principal $\overline{U}$ from $\mathcal{U}$. $\overline{U}$ is $\mathcal{F}$'s guess at which principal $\mathcal{A}$ will choose for the forgery. The adversary assigns $\overline{e}$ as the public key of $\overline{U}$. For all other principals, $\mathcal{F}$ generates the signing keys using the signature key generation algorithm $\overline{\mathcal{K}}$. $\mathcal{F}$ also generates the encryption keys for all the principals using $\mathcal{K}$. This allows $\mathcal{F}$ to answer all the oracle queries from $\mathcal{A}$ as follows.

$\mathsf{Send}(U, s, m)$ Assume $m = \mathsf{Initiator}$. According to the protocol specification, a random nonce is generated and encrypted under the keys of the responders. Since all encryption keys are known, all the ciphertexts can be formed. If

$U \neq \overline{U}$ then the signing key is available too, otherwise the signature is obtained by querying the signing oracle. The ciphertexts and signature are returned to the adversary. If $m \neq \mathsf{Initiator}$ then this query can be answered normally as per protocol specification.

$\mathsf{Reveal}(U, s)$ Since all the session keys are known from the $\mathsf{Send}(U, s, m)$ queries, the query can be trivially answered with the correct session key (if available).

$\mathsf{Corrupt}(U, K)$ As long as $U \neq \overline{U}$ all the private information is available and the query can be answered. In the case $U = \overline{U}$ then the query cannot be properly answered, and fails.

$\mathsf{Test}(U, s)$ Since all the accepted session keys are known from running the send queries, the query can be trivially answered by identifying the correct session key.

If during the execution of $\mathcal{A}$, $\mathcal{A}$ makes a query that includes a forged signature, then $\mathcal{F}$ returns the forgery and halts. Otherwise, $\mathcal{F}$ halts when $\mathcal{A}$ does and outputs fail. Notice that when a $\mathsf{Corrupt}$ query fails, this means that the guess of $\overline{U}$ as the user whose signature was to be forged by $\mathcal{A}$ was wrong. (Recall that we are assuming that $\mathcal{A}$ gets her advantage by forging a signature.) Suppose that $\mathcal{A}$ succeeds by forging a signature with probability at least $\nu_s(k)$. The probability that this is a forgery for $\overline{e}$ is at least $1/n$. Therefore the signature forger succeeds with probability

$$\mathsf{Succ}_{\Sigma}(k) \geq \nu_s(k)/n.$$

In other words, the success probability of an adversary attacking the protocol in this case is at most n times the probability of signature forgery. Since n is polynomial in the security parameter, k, if $\nu_s(k)$ is non-negligible, then so is $\mathsf{Succ}_{\Sigma}(k)$.

5.2 Encryption attacker

Now assume that $\mathcal{A}$ gains an advantage without forging a signature. This time we use $\mathcal{A}$ to form an algorithm $\mathcal{X}$ which has an advantage against the underlying encryption scheme $\mathcal{PE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ in the multi-user setting.

The input to $\mathcal{X}$ consists of the following.

- Public keys $e_2, \ldots e_n$ generated by $\mathcal{K}(1^k)$.
- Two randomly chosen values σ_0 and σ_1 of equal bit length k.
- Encryptions $\alpha_2 = \mathcal{E}_{e_2}(\sigma_\theta), \ldots, \alpha_n = \mathcal{E}_{e_n}(\sigma_\theta)$, where θ is randomly chosen in $\{0, 1\}$.

The goal of $\mathcal{X}$ is to gain an advantage in guessing whether $\theta = 0$ or $\theta = 1$.

Algorithm $\mathcal{X}$ runs as follows. Firstly, $\mathcal{X}$ chooses at random a principal from $\mathcal{U}$. Without loss of generality we assume this principal to be U_1. $\mathcal{X}$ then proceeds to distribute long-lived keys to all principals. $\mathcal{X}$ assigns signature keys $(\overline{e}_i, \overline{d}_i) \leftarrow \overline{\mathcal{K}}(1^k)$ to each user U_i for $i \in [1, n]$. In order to distribute encryption keys, $\mathcal{X}$ assigns $(e_1, d_1) \leftarrow \mathcal{K}(1^k)$ to U_1 and the public encryption keys $e_2, \ldots, e_n$ to principals $U_2, \ldots, U_n$, respectively. $\mathcal{X}$ also chooses a random session identifier, $t \in [1, S]$, where S is the maximum number of sessions that the adversary

$\mathcal{A}$ is allowed to instantiate. S is polynomial in the security parameter k. The identifier t is used to decide when the initiator will give the input ciphertexts to $\mathcal{A}$. Algorithm $\mathcal{X}$ answers all the oracle queries from $\mathcal{A}$ as follows.

$\mathsf{Send}(U, s, m)$ If the query is to start a new protocol run, we have the following two cases.

1. Suppose that (U, s) is to be the initiator of the protocol. If $s = t$ and $U = U_1$, then $\alpha_2, \ldots, \alpha_n$ are used as the encrypted values for the other $n-1$ principals and signed using U_1's signing key. The ciphertexts $\alpha_2, \ldots, \alpha_n$ and signature are returned to the adversary.
   Otherwise, a nonce is chosen randomly of k bits, and is then encrypted under the public encryption keys of the rest of the principals. Since all encryption keys are known, all the ciphertexts can be formed. Similarly, since all signing keys are available to $\mathcal{X}$ the signature of U on these encryptions can also be computed. The ciphertexts and signature are returned to $\mathcal{A}$.
   Algorithm $\mathcal{X}$ should record that (U, s) is the initiator and the nonce and ciphertext values chosen.
2. Now suppose that (U, s) is the responder. Then a nonce is chosen randomly of k bits. The nonce is returned to the adversary. Algorithm $\mathcal{X}$ should record that (U, s) is the responder and the nonce value chosen.

If the query is not to start a new protocol run, then we get the following two cases.

1. If (U, s) is the initiator, then Π_U^s accepts a new conference key provided m is of the expected form (a set of $n-1$ nonces of the correct length); otherwise, it fails. The outcome of whether it accepts or fails is returned to $\mathcal{A}$. Algorithm $\mathcal{X}$ records the nonces received by (U, s).
2. If (U, s) is a responder then m must consist of a signature, $n-1$ ciphertexts and $n-2$ nonces of correct size. If the format is correct and the signature is verified correctly then Π_U^s accepts and this information is returned to $\mathcal{A}$. Otherwise it outputs $'\mathsf{fail}'$. Algorithm $\mathcal{X}$ records the signature, ciphertexts and nonces received by (U, s).

$\mathsf{Reveal}(U, s)$ Since session keys are modelled as the output of a random oracle, we only need to keep track of which keys have been revealed before. If a key has not been revealed then a random string is returned. If it has been revealed then the same value as used before is returned. Let us assume Π_U^s has accepted (otherwise the query fails).
If (U, s) is a responder then it must have received and accepted signed ciphertexts. Since the adversary cannot forge signatures, the ciphertexts must have been formed in a Send query and are known to $\mathcal{X}$. (They were either formed by $\mathcal{X}$ or they are the α_i values.) If (U, s) is an initiator then $\mathcal{X}$ knows which nonces were received by (U, s) and also which ciphertexts were output by $(U, s$ to intitiate the protocol run. Therefore, in either case, $\mathcal{X}$ knows if the key have been revealed before and can respond correctly.

$\mathsf{Corrupt}(U, K)$ As long as $U = U_1$ then all the private information is available and the query can be answered. Otherwise the query cannot be answered and $\mathcal{A}$ will fail.

$\mathsf{Test}(U, s)$ If $s \neq t$ or the initiator of the conference for session s is not U_1 then the algorithm fails. Otherwise, $\mathcal{X}$ outputs a random string.

At some stage, $\mathcal{A}$ completes and returns a value b. We need to show how the prediction of the session key allows prediction of the plaintext chosen as input for $\mathcal{X}$. To do this we make use of the random oracle to model the hash function. Therefore we assume that whenever $\mathcal{A}$ makes a query of the hash function the value is returned randomly except if the same query was previously asked. The oracle keeps a list of all previously asked queries, and if the same query is asked again then the response is the same as the first time. Because of the random output of the oracle, $\mathcal{A}$ can gain no advantage in guessing any key for which the oracle is not queried. Therefore $\mathcal{X}$ examines all queries made by $\mathcal{A}$ of the form $h(N_1, N_2, \ldots, N_n)$. If there exists a query with $N_1 = \sigma_0$ then $\mathcal{X}$ returns $\theta = 0$. Otherwise it returns $\theta = 1$. Let $\nu_e(k)$ be the success probability of $\mathcal{A}$. Since the probability that during $\mathcal{X}$ execution the test query does not fail is $\frac{1}{nS}$, $\mathcal{X}$'s success probability is

$$\mathsf{Succ}_{n\mathcal{PE}}(k) \geq \frac{\nu_e(k)}{nS}.$$

Thus, on the assumption that $\mathcal{A}$ does not perform a signature forgery, we have shown that a non-negligible (in k) advantage in attacking the indistinguishability property of the conference key agreement scheme can be turned into a non-negligible advantage to attack the encryption scheme in the multiple-user setting, and by virtue of Lemma 1 into a non-negligible advantage to attack it in the single-user setting.

6 Conclusion

We have described the first known conference key agreement protocol that can be completed in one round and provided a proof of its security. It remains an open question whether it is possible to design a multi-party contributory key agreement scheme which completes in one round of communication and also provides forward security.

References

1. Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. Authenticated group key agreement and friends. In *5th Conference on Computer and Communications Security*, pages 17–26. ACM Press, 1998.
2. Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. New multi-party authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4):628–639, April 2000.
3. Klaus Becker and Uta Wille. Communication complexity of group key distribution. In *5th Conference on Computer and Communications Security*, pages 1–6. ACM Press, 1998.

4. M. Bellare and P. Rogaway. Provably secure session key distribution – the three party case. In *Proceedings of the 27th ACM Symposium on the Theory of Computing*, 1995.
5. Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In B. Preneel, editor, *Advances in Cryptology – Eurocrypt 2000*, volume 1807 of *LNCS*. Springer-Verlag, 2000. Full version at `http://www-cse.ucsd.edu/users/mihir/papers/key-distribution.html`.
6. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology - Eurocrypt 2000*, pages 139–155. Springer-Verlag, 2000.
7. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Advances in Cryptology – CRYPTO'93*, pages 232–249. Springer-Verlag, 1993. Full version at `www-cse.ucsd.edu/users/mihir`.
8. S. Blake-Wilson and A. Menezes. Security proofs for entity authentication and authenticated key transport protocols employing asymmetric techniques. In *Security Protocols Workshop*. Springer-Verlag, 1997.
9. Simon Blake-Wilson and Alfred Menezes. Authenticated Diffie-Hellman key agreement protocols. In *Selected Areas in Cryptography*, pages 339–361. Springer-Verlag, 1999.
10. Victor Boyko, Philip MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *Advanced in Cryptology - Eurocrypt 2000*. Springer-Verlag, 2000.
11. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Provably authenticated group Diffie-Hellman key exchange – the dynamic case. In *Advances in Cryptology - Asiacrypt 2001*, pages 290–309. Springer-Verlag, 2001.
12. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In *Advances in Cryptology - Eurocrypt 2002*. Springer-Verlag, 2002.
13. Emmanuel Bresson, Olivier Chevassut, David Pointcheval, and Jean-Jacques Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *CCS'01*, pages 255–264. ACM Press, November 2001.
14. Mike Burmester and Yvo Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology – Eurocrypt'94*, pages 275–286. Springer-Verlag, 1995.
15. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transaction on Information Theory*, 22:644–654, 1976.
16. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer Security*, 28:270–299, 1984.
17. Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2), 1988.
18. Ingemar Ingemarsson, Donald T. Tang, and C.K.Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, IT-28(5):714–720, September 1982.
19. Antoine Joux. A one round protocol for tripartite Diffie-Hellman. In W. Bosma, editor, *Algorithmic Number Theory, 4th International Symposium, ANTS-IV*, volume 1838 of *LNCS*, pages 385–393. Springer-Verlag, 2000.
20. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

21. Olivier Pereira and Jean-Jacques Quisquater. A security analysis of the Cliques protocol suites. In *Computer Security Foundations Workshop*, pages 73–81. IEEE Computer Society Press, 2001.
22. Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer-Verlag, Berlin, Germany, 1992.
23. Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman key distribution extended to group communication. In *3rd ACM Conference on Computer and Communications Security*, New Delhi, March 1996. ACM Press.
24. Wen-Guey Tzeng and Zhi-Jha Tzeng. Round-efficient conference key agreement protocols with provable security. In T. Okamoto, editor, *Advances in Cryptology – Asiacrypt 2000*, volume 1976 of *LNCS*, pages 614–627. Springer-Verlag, 2000.

A Attack on Tzeng and Tzeng's Second Protocol

Tzeng and Tzeng's second protocol [24] relies on a proof of knowledge to provide authentication of the principals. In this appendix we show that this proof is not sound so that the protocol provides no authentication.

In the conference key protocol a non-interactive version of the proof is used, but to explain the idea the authors use a conventional interactive description which we will review now. The prover is one member of the conference which has n users with public keys y_j for $1 \leq j \leq n$. The prover has public key $\tilde{y}$ and corresponding private key $\tilde{x}$ so that $\tilde{y} = g^{\tilde{x}}$. The prover chooses $k \in_R \mathbb{Z}_q$ and calculates $u_j = y_j^k$ for $1 \leq j \leq n$.

$$
\begin{array}{rcl}
P & & V \\
r_1, r_2 \in_R \mathbb{Z}_q & & \\
b_j = y_j^{r_1} g^{r_2}, 1 \leq j \leq n & & \\
 & \xrightarrow{b_1, b_2, \ldots, b_n} & \\
 & & c \in_R [0..2^t - 1] \\
 & \xleftarrow{c} & \\
w_1 = r_1 - ck & & \\
w_2 = r_2 - c\tilde{x} & \xrightarrow{w_1, w_2} & b_j \stackrel{?}{=} y_j^{w_1} g^{w_2} (\tilde{y} u_j)^c
\end{array}
$$

The purpose of the proof is to show that:

- the values $\log_{y_j} u_j$ are equal for all j and known to P.
- P knows the secret $\tilde{x}$.

We now show that an adversary A can masquerade as the prover without knowing the value $\tilde{x}$. First A chooses $v_j \in_R \mathbb{Z}_q$ and chooses the values u_j by solving $u_j \tilde{y} = g^{v_j}$. (In the protocol the u_j values are sent together with the non-interactive proof.) Then A can choose the commitments in the usual way as $b_j = y_j^{r_1} g^{r_2}$ for randomly chosen r_1, r_2. Now when A receives the challenge c he can compute $w_1 = r_1$ and $w_2 = -v_j c + r_2$ and the checks by V will succeed.